

DA RESTITUIRE INSIEME AGLI ELABORATI e A TUTTI I FOGLI
--

SVOLGIMENTO DELLA PROVA:

PER GLI STUDENTI DI "ARCHITETTURA DEI CALCOLATORI – A.A. 2015/16": es. N.1 + es. N.3 + es. N.4

PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere sia il modulo CALCOLATORI che il modulo RETI: es. N.1,2,3,5

PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere SOLO il modulo CALCOLATORI es. N.1,2,3.

PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere SOLO il modulo RETI: es. N.4,5

NOTA: per l'esercizio1 (e analogamente per l'esercizio 4) dovranno essere consegnati due files: il file del programma MIPS (ovvero VERILOG) e il file relativo all'output (screenshot o copy/paste)

1. [18] Utilizzando il simulatore SPIM, codificare in assembly MIPS il seguente codice (**utilizzando solo e unicamente istruzioni dalla tabella sottostante**), **rispettando le convenzioni di utilizzazione dei registri dell'assembly MIPS** (riportate in calce). Al termine della codifica consegnare 2 files: il programma in MIPS e l'output relativo.

```

#define N 4
#define MAXITER 20

double A[N*N]={10.0, -1.0, 2.0, 0.0, -1.0, 11.0, -1.0,
               3.0, 2.0, -1.0, 10.0, -1.0, 0.0, 3.0, -1.0, 8.0};

double b[N]={6.0, 25.0, -11.0, 15.0};
double x[N]={0.0, 0.0, 0.0, 0.0};
double y[N], c[N], R[N*N];
int iter = 0;
double e;

void seidel(double *x, double *y, double *a, double c) {
    int j;
    *x = c;
    for (j = 0; j < N; ++j) *x -= a[j] * y[j];
}

void setup() {
    int i, j;
    for (i = 0; i < N; ++i) {
        c[i] = b[i] / A[i+i*N];
        for (j = 0; j < N; ++j) {
            R[i*N+j] = (i==j) ? 0 : A[i*N+j] / A[i*N+i];
        }
    }
}

int report() {
    int i;
    print_string("X: ");
}

for (i = 0; i < N; ++i) {
    print_double(x[i]); print_string(" ");
}
print_string(" - iter="); print_int(iter);
print_string(" e="); print_double(e);
print_string("\n");
}

int test_convergence() {
    int j, r;
    ++iter; e = 0;
    for (j = 0; j < N; ++j)
        e += (y[j] - x[j]) * (y[j] - x[j]);
    r = (e < 0.0001 * 0.0001 || iter == MAXITER);
    report();
    return (r);
}

int compute() {
    int i;
    do {
        for (i = 0; i < N; ++i) y[i] = x[i];
        for (i = 0; i < N; ++i) seidel(&x[i], x, &R[N*i], c[i]);
    } while (!test_convergence());
}

int main()
{
    setup(); compute(); report(); exit(0);
}

```

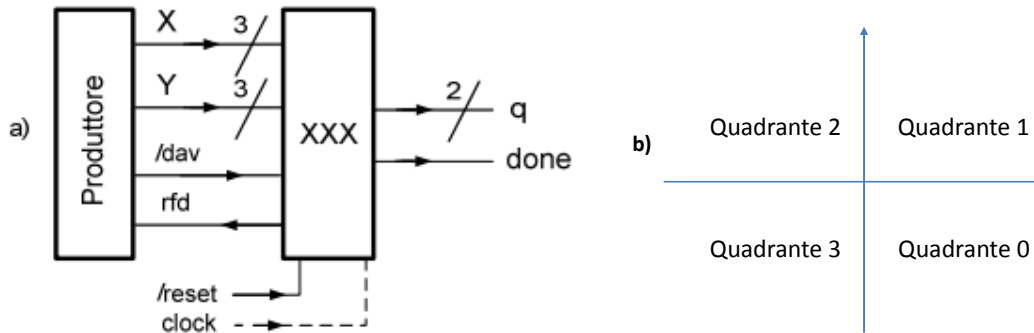
2. [7] Si consideri una cache di dimensione 128B e a 4 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 16 byte, il tempo di accesso alla cache e' 4 ns e la penalita' in caso di miss e' pari a 40 ns, la politica di rimpiazzamento e' LRU. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 124, 169, 167, 245, 183, 119, 235, 163, 288, 309, 310, 308, 213, 196, 377, 166, 362, 233, 163, 169. Tali accessi sono alternativamente letture e scritture. Per la sequenza data, ricavare il tempo medio di accesso alla cache, riportare i tag contenuti in cache al termine, i bit di modifica (se presenti) e la lista dei blocchi (ovvero il loro indirizzo) via via eliminati durante il rimpiazzamento ed inoltre in corrispondenza di quale riferimento il blocco e' eliminato.
3. [5] In un processore MIPS con pipeline determinare i cicli necessari per eseguire due iterazioni per il seguente frammento di codice sia nel caso di propagazione abilitata che di propagazione disabilitata. Nota: e' presente 1 delay-slot e l'accesso ai registri nella fase di decodifica e write-back possono essere sovrapposte.

```

    add $1, $2, $3
L1: lw  $4,0($1)
    add $6, $1, $4
    lw  $5,0($1)
    bne $6, $5, L1
    nop

```

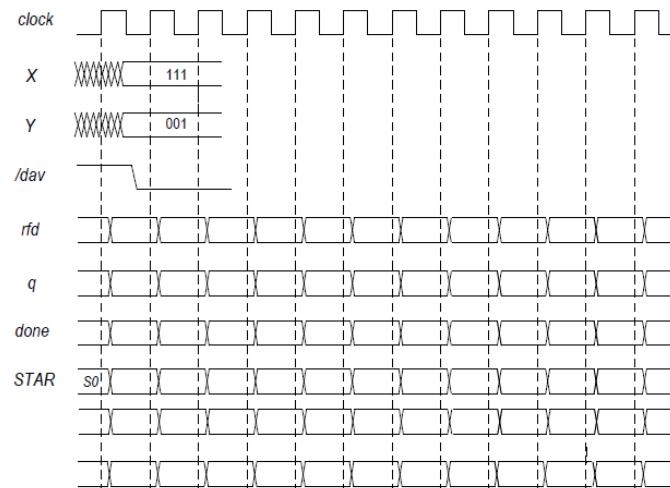
4) [10] L'unità XXX (Fig. a) preleva dal Produttore i due numeri naturali X e Y e li utilizza come le rappresentazioni (in complemento a due) di due numeri interi x e y. Interpreta x e y come le coordinate di un punto nel piano cartesiano ed emette tramite l'uscita q il numero d'ordine (0, 1, 2, 3) del quadrante a cui appartiene il punto (Fig. b) tenendo done ad 1 per un ciclo di clock.



Si specifichino i numeri d'ordine dei quadranti in modo da ottimizzare la rete combinatoria che produce q e si riportino tali numeri d'ordine nella Fig. b. Sempre in questa ottica, si considerino i semiassi come opportunamente appartenenti ai quadranti. Per chiarirsi le idee è utile completare preliminarmente la seguente tabella

x	y	X	Y	q
+1	+1	001	001	1
+1	-1			
-1	-1			
-1	+1			
+0	+0	000	000	1
+0	+1			
+0	-1			
+1	+0			
-1	+0			

Si descriva e si sintetizzi l'unità XXX e se ne tracci l'evoluzione nell'ipotesi che il Produttore fornisca le rappresentazioni di $(x,y) = (-1,+1), (0,+2), (1,3), (2,-4), (3,-3), (-4,-2)$ indicando chiaramente nel grafico tali rappresentazioni (il codice Verilog del produttore e del top-module e' fornito nell'ultima pagina del testo).



5) [12] Sintetizzare un riconoscitore di sequenze 11-01-10 utilizzando il modello di Moore. Rappresentare la macchina a stati finiti per tale riconoscitore, la tabella delle transizioni, le equazioni booleane delle reti CN1 e CN2 e il circuito sequenziale sincronizzato basato su flip-flop D.

Instructions

Instruction	Example	Meaning	Comments
add	add \$1,\$2,\$3	\$1 = \$2 + \$3	3 operands; exception possible
subtract	sub \$1,\$2,\$3	\$1 = \$2 - \$3	3 operands; exception possible
add immediate	addi \$1,\$2,100	\$1 = \$2 + 100	+ constant; exception possible
subtract immediate	subi \$1,\$2,100	\$1 = \$2 - 100	- constant; exception possible
multiplication	mult \$1,\$2	Hi,Lo= \$1 x \$2	64-bit Signed Product ; result in Hi,Lo
division	div \$1,\$2	Hi= \$1 % \$2, Lo = \$1 / \$2	Signed division
move from Hi	mghi \$1	\$1 = Hi	Create copy of Hi
move from Lo	mflr \$1	\$1 = Lo	Create copy of Lo
and	and \$1,\$2,\$3	\$1 = \$2 & \$3	3 register operands; Logical AND
or	or \$1,\$2,\$3	\$1 = \$2 \$3	3 register operands; Logical OR
nor	nor \$1,\$2,\$3	\$1 = !(\$2 \$3)	3 register operands; Logical NOR
xor	xor \$1,\$2,\$3	\$1 = \$2 ^ \$3	3 register operands; Logical XOR
and immediate	andi \$1,\$2,100	\$1 = \$2 & 100	Logical AND register, constant
or immediate	ori \$1,\$2,100	\$1 = \$2 100	Logical OR register, constant
xor immediate	xori \$1,\$2,100	\$1 = \$2 ^ 100	Logical XOR register, constant
shift left logical	sll \$1,\$2,10	\$1 = \$2 << 10	Shift left by constant
shift right logical	srl \$1,\$2,10	\$1 = \$2 >> 10	Shift right by constant
load word	lw \$1,100(\$2)	\$1 = Memory[\$2+100]	Data from memory to register
load byte	lb \$1,100(\$2)	\$1 = Memory[\$2+100]	Data from memory to register
load byte unsigned	lbu \$1,100(\$2)	\$1 = Memory[\$2+100]	Data from mem. to reg.; no sign extension
store word	sw \$1,100(\$2)	Memory[\$2+100] = \$1	Data from register to memory
store byte	sb \$1,100(\$2)	Memory[\$2+100] = \$1	Data from register to memory
load address	la \$1,var	\$1 = &var	Load variable address
branch unconditional	b 100	go to PC+4+100	PC relative branch
branch on equal	beq \$1,\$2,100	if (\$1 == \$2) go to PC+4+100	Equal test; PC relative branch
branch on not equal	bne \$1,\$2,100	if (\$1 != \$2) go to PC+4+100	Not equal test; PC relative
set on less than	slt \$1,\$2,\$3	if (\$2 < \$3) \$1 = 1; else \$1 = 0	Compare less than; 2's complement
set on less than immediate	slti \$1,\$2,100	if (\$2 < 100) \$1 = 1; else \$1 = 0	Compare < constant; 2's complement
set on less than unsigned	sltu \$1,\$2,\$3	if (\$2 < \$3) \$1 = 1; else \$1 = 0	Compare less than; natural number
set on less than imm.unsigned	sltiu \$1,\$2,100	if (\$2 < 100) \$1 = 1; else \$1 = 0	Compare constant; natural number
jump	j 10000	go to 10000	Jump to target address
jump register	jr \$31	go to \$31	For switch, procedure return
jump and link	jal 10000	\$31 = PC + 4; go to 10000	For procedure call
add.s add.d	add.x \$f0,\$f2,\$f4	\$f0=\$f2+\$f4	Single and double precision add
sub.s sub.d	add.x \$f0,\$f2,\$f4	\$f0=\$f2-\$f4	Single and double precision subtraction
mul.s mul.d	mul.x \$f0,\$f2,\$f4	\$f0=\$f2*\$f4	Single and double precision multiplication
div.s div.d	div.x \$f0,\$f2,\$f4	\$f0=\$f2/\$f4	Single and double precision division
mov.s mov.d	mov.x \$f0,\$f2	\$f0←\$f2	Single and double precision move
abs.s abs.d	abs.x \$f0,\$f2	\$f0=ABS(\$f2)	Single and double precision absolute value
neg.s neg.d	neg.x \$f0,\$f2	\$f0= - (\$f2)	Single and double precision opposite value
c.lt.s c.lt.d (eq,ne,le,gt,ge)	c.lt.x \$f0,\$f2	Temp=(\$f0<\$f2)	Single and double: compare \$f0 and \$f2 <,<=,>,>=
mtc1	mtc1 \$1,\$f2	\$f2=\$1	Data from gen.reg. \$1 to C1 reg. \$f2 (no conversion)
mfc1	mfc1 \$f1,\$1	\$1=\$f2	Data from gen.reg. to C1 reg. (no conversion)
branch on false	bc1f label	If (Temp == false) go to label	Temp is 'Condition-Code'
branch on true	bc1t label	If (Temp == true) go to label	Temp is 'Condition-Code'
load floating point (32bit)	lwc1 \$f0,0(\$1)	\$f0←Memory[\$1]	Data from FP (C1) register to memory
store floating point (32bit)	swc1 \$f0,0(\$1)	Memory[\$1]←\$f0	Data from memory to FP (C1) register
convert single into double	cvt.d.s \$f0,\$f2	\$f0=(double)\$f2	Also cvt.s.d (viceversa)
convert single into integer	cvt.w.s \$f1,\$f0	\$f1=(int)\$f0	Also cvt.s.w (viceversa)

Register Usage

Name	Reg. Num.	Usage	Name	Reg. Num.	Usage	Reg. Num.	Usage
\$zero	0	The constant value 0	\$v0-\$v1	2-3	Results	\$f0, \$f2	Return values
\$s0-\$s7	16-23	Saved	\$fp, \$sp	30,29	frame pointer, stack pointer	\$f12,\$f14	Function arguments
\$t0-\$t9	8-15,24-25	Temporaires	\$ra, \$gp	31,28	return address, global pointer	\$f20,\$f22,\$f24, \$f26,\$f28,\$f30	Saved registers
\$a0-\$a3	4-7	Arguments	\$k0-\$k1	26,27	Kernel usage	\$f4,\$f6,\$f8,\$f10, \$f16,\$f18	Temporaries registers

System calls

Service Name	Service Num. (\$v0)	INPUT Arguments	OUTPUT Arguments
print_int	1	\$a0=integer to print	---
print_float	2	\$f12=float to print	---
print_double	3	(\$f12,\$f13)=double to print	---
print_string	4	\$a0=address of ASCIIZ string to print	---
read_int	5	---	\$v0=integer
read_float	6	---	\$f0=float
read_double	7	---	\$f0-f1=double
read_string	8	\$a0=address of input buffer, \$a1=max characters to read	---
sbrk	9	\$a0=Number of bytes to be allocated	\$v0=pointer to the allocated memory
exit	10	---	---

```

module TopLevel;
  reg reset_; initial begin reset_=0; #1 reset_=1; #300; $stop; end
  reg clock ; initial clock =0; always #5clock <=(!clock);
  wire[2:0] X,Y;
  wire rfd, dav_;
  wire[1:0]q; wire done;
  wire[1:0] STAR=Xxx.STAR;
  XXX Xxx(rfd, dav_,X,Y, q,done, clock,reset_);
  Produttore PRO(rfd,dav_,X,Y);
endmodule

```

```

module Produttore(rfd,dav_,X,Y);
  input      rfd;
  output     dav_;
  output [3:0] X,Y;
  reg DAV_;  assign dav_=DAV_;
  reg [2:0]  APP1_X, APP2_X, APP1_Y, APP2_Y; assign X=APP1_X, Y=APP1_Y;
  initial begin APP2_X='B111; APP2_Y=001; DAV_=1; end
  always
    begin #5; wait(rfd==1); #3 APP1_X=APP2_X; APP2_X=APP2_X+1;
APP1_Y=APP2_Y; APP2_Y=APP2_Y+1;
          #5 DAV_=0; wait(rfd==0); #1 APP1_X='HXX; APP1_Y='HXX; #9
DAV_=1;end
endmodule

```

COMPITINO di ARCHITETTURA DEI CALCOLATORI del 25-01-2016
 SOLUZIONE - versione 1.2

ESERCIZIO 1)

```
.data
A: .double 10.0, -1.0, 2.0, 0.0
   .double -1.0, 11.0, -1.0, 3.0
   .double 2.0, -1.0, 10.0, -1.0
   .double 0.0, 3.0, -1.0, 8.0
bv: .double 6.0, 25.0, -11.0, 15.0
x:  .double 0.0, 0.0, 0.0, 0.0
y:  .space 32
c:  .space 32
R:  .space 128
iter:.word 0
e:  .space 8
spazio: .asciiz " "
ret:   .asciiz "\n"
iter_str: .asciiz " - iter="
e_str:  .asciiz " e="
x_str:  .asciiz "X: "
eps:   .double 0.0001

.text
.globl main

main:
# _____ NO CALL FRAME _____
jal setup
jal compute
jal report
addi $v0, $0, 10
syscall

setup:
# _____ NO CALL FRAME _____
la $t5, c # c
la $t6, bv # &b
la $t7, A # &A
la $t8, R # &R

add $t2, $0, $0 # t2=i=0
Setup_start_for_1:
slti $t4, $t2, 4 # i <? N(4)
beq $t4, $0, Setup_end_for_1
# _____
sll $t0, $t2, 3 # t4= i*sizeof(double)
add $t4, $t0, $t6 # t4= &b[i]
lwc1 $f4, 0($t4)
lwc1 $f5, 4($t4) # f4:f5=b[i]
addi $t1, $0, 4 # N
mult $t2, $t1
mflo $t1 # t1=i*N
add $t4, $t1, $t2 # t4=i*N+i
sll $t4, $t4, 3 # *8
add $t4, $t4, $t7 # t4= &A[i*N+i]
lwc1 $f6, 0($t4)
lwc1 $f7, 4($t4) # f6:f7=A[i*N+i]
div.d $f8, $f4, $f6 # b[i]/A[i*N+i]
add $t4, $t0, $t5 # t4=&c[i]
swc1 $f8, 0($t4)
swc1 $f9, 4($t4) # store c[i]
add $t3, $0, $0 # t3=j=0
Setup_start_for_2:
slti $t4, $t3, 4 # j <? N(4)
beq $t4, $0, Setup_end_for_2
# _____
add $t0, $t1, $t3 # t0= i*N+j
sll $t0, $t0, 3 # *8
add $t9, $t0, $t8 # t9= &R[i*N+j]
bne $t2, $t3, Setup_else
sw $0, 0($t9)
sw $0, 4($t9)
j Setup_fi
Setup_else:
add $t4, $t1, $t2 # t4=i*N+i
sll $t4, $t4, 3 # *8
add $t4, $t4, $t7 # t4= &A[i*N+i]
lwc1 $f6, 0($t4)
lwc1 $f7, 4($t4)
add $t4, $t0, $t7 # t4= &A[i*N+j]
lwc1 $f4, 0($t4)

lwc1 $f5, 4($t4)
div.d $f8, $f4, $f6 # f8=A[N*i+j]/A[N*i,i]
swc1 $f8, 0($t9)
swc1 $f9, 4($t9)

Setup_fi:
# _____
addi $t3, $t3, 1
j Setup_start_for_2
Setup_end_for_2:
# _____
addi $t2, $t2, 1
j Setup_start_for_1
Setup_end_for_1:
jr $ra

compute:
# _____ CALL FRAME _____
# saved variables: s0 4B
# ra 4B
# _____ Totale 8B _____
addi $sp, $sp, -8
sw $s0, 4($sp)
sw $ra, 0($sp)
Compute_start_do:
add $s0, $0, $0 # s0=i=0
Compute_start_for_1:
slti $t9, $s0, 4 # i <? N
beq $t9, $0, Compute_end_for_1
# _____
sll $t3, $s0, 3 # t3=i*8
la $a0, y
add $a0, $a0, $t3 # a0=&y+i*8
la $a1, x
add $a1, $a1, $t3 # a0=&x+i*8
lwc1 $f12, 0($a1) # load x[i]
lwc1 $f13, 4($a1)
swc1 $f12, 0($a0) # store y[i]
swc1 $f13, 4($a0)
# _____
addi $s0, $s0, 1
j Compute_start_for_1
Compute_end_for_1:
add $s0, $0, $0 # s0=i=0
Compute_start_for_2:
slti $t9, $s0, 4 # i <? N
beq $t9, $0, Compute_end_for_2
# _____
sll $t3, $s0, 3 # t3=i*8
la $a0, x
add $a0, $a0, $t3 # a0=&x+i*8
la $a1, x
addi $t1, $0, 4 # N=4
mult $t3, $t1 # N*i*8
mflo $t0 #
la $a2, R
add $a2, $a2, $t0 # a2=R+N*i*8
la $t4, c # t4=&c[0]
add $t4, $t4, $t3 # t4=&c[i]
lwc1 $f12, 0($t4)
lwc1 $f13, 4($t4)
jal seidel
# _____
addi $s0, $s0, 1
j Compute_start_for_2
Compute_end_for_2:
jal test_Convergence # Output:v0
beq $v0, $0, Compute_start_do

Compute_end_do:
lw $s0, 4($sp)
lw $ra, 0($sp)
addi $sp, $sp, 8
jr $ra
```

COMPITINO di ARCHITETTURA DEI CALCOLATORI del 25-01-2016
 SOLUZIONE - versione 1.2

```

report:
# _____CALL FRAME_____
# saved variables: s0      4B
# ra                      4B
# _____Totale  8B_____
addi $sp, $sp, -8
sw $s0, 4($sp)
sw $ra, 0($sp)

addi $v0, $0, 4      # print "X: "
la $a0, x_str
syscall
add $s0, $0, $0      # s0=j=0
Report_start_for_1:
  slti $t9, $s0, 4   # j <? N
  beq $t9, $0, Report_end_for_1
  # _____
  addi $t2, $0, 8
  mult $t2, $s0
  mflo $t3            # t3=i*8
  la $t4, x
  add $t4, $t4, $t3   # t4= x[i]
  lwcl $f12, 0($t4)
  lwcl $f13, 4($t4)
  addi $v0, $0, 3     # print x[i]
  syscall
  addi $v0, $0, 4     # print " "
  la $a0, spazio
  syscall
  # _____
  addi $s0, $s0, 1
  j Report_start_for_1
Report_end_for_1:
addi $v0, $0, 4      # print " - iter="
la $a0, iter_str
syscall
addi $v0, $0, 1      # print iter
la $t0, iter
lw $a0, 0($t0)
syscall
addi $v0, $0, 4      # print " e="
la $a0, e_str
syscall
addi $v0, $0, 3
la $t0, e            # print e
lwcl $f12, 0($t0)
lwcl $f13, 4($t0)
syscall
addi $v0, $0, 4      # print "\n"
la $a0, ret
syscall

lw $s0, 4($sp)
lw $ra, 0($sp)
addi $sp, $sp, 8
jr $ra

seidel:
# _____NO CALL FRAME_____
# INPUT: a0 = *x, a1 = *y, a2 = *a, f12:f13 = c

swcl $f12, 0($a0)    # store c into *x
swcl $f13, 4($a0)

add $t0, $0, $0      # t0=j=0
Seidel_start_for_1:
  slti $t9, $t0, 4   # N <? 4
  beq $t9, $0, Seidel_end_for_1
  # _____
  sll $t3, $t0, 3    # j*8
  add $t4, $t3, $a2  # &a[j]
  lwcl $f4, 0($t4)
  lwcl $f5, 4($t4)
  add $t4, $t3, $a1  # &y[j]
  lwcl $f6, 0($t4)
  lwcl $f7, 4($t4)
  mul.d $f8, $f4, $f6 # a[j]*y[j]
  lwcl $f10, 0($a0)  # load *x
  lwcl $f11, 4($a0)  # load *x
  sub.d $f10, $f10, $f8 # *x-=
  swcl $f10, 0($a0)  # store *x
  swcl $f11, 4($a0)
  # _____
  addi $t0, $t0, 1
  j Seidel_start_for_1
Seidel_end_for_1:
jr $ra

test_Convergence:
# _____CALL FRAME_____
# ra                      4B
# _____Totale  4B_____
addi $sp, $sp, -4
sw $ra, 0($sp)

la $t0, iter
lw $t1, 0($t0)
addi $t1, $t1, 1     # ++iter
sw $t1, 0($t0)

la $t0, e
sw $0, 0($t0)
sw $0, 4($t0)        # e=0

add $t0, $0, $0      # t0=j=0
Test_Convergence_start_for:
  slti $t9, $t0, 4   # j <? N
  beq $t9, $0, Test_Convergence_end_for
  # _____
  la $t2, x
  la $t3, y
  sll $t5, $t0, 3    # j*8
  add $t2, $t2, $t5  # &x[j]
  lwcl $f4, 0($t2)
  lwcl $f5, 4($t2)
  add $t3, $t3, $t5  # &y[j]
  lwcl $f6, 0($t3)
  lwcl $f7, 4($t3)
  sub.d $f8, $f6, $f4 # y-x
  mul.d $f8, $f8, $f8 # ()^2
  la $t6, e
  lwcl $f10, 0($t6)  # load e
  lwcl $f11, 4($t6)
  add.d $f10, $f10, $f8 # e+=()
  swcl $f10, 0($t6)  # store e
  swcl $f11, 4($t6)
  # _____
  addi $t0, $t0, 1
  j Test_Convergence_start_for
Test_Convergence_end_for:

jal report

la $t0, eps
lwcl $f4, 0($t0)
lwcl $f5, 4($t0)
mul.d $f4, $f4, $f4 # eps * eps

c.lt.d $f10, $f4     # e < eps
add $v0, $0, $0     # r=0
bclf Test_Convergence_c1_f
  addi $v0, $0, 1
  j Test_Convergence_c2_end
Test_Convergence_c1_f:
  la $t0, iter
  lw $t0, 0($t0)
  slti $t2, $t0, 20 # iter ==? MAXITER(20)
  bne $t2, $0, Test_Convergence_c2_end
  addi $v0, $0, 1

Test_Convergence_c2_end:
lw $ra, 0($sp)
addi $sp, $sp, 4
jr $ra

```

COMPITINO di ARCHITETTURA DEI CALCOLATORI del 25-01-2016
 SOLUZIONE - versione 1.2

ESERCIZIO 2)

A=4,B=16,C=128.

Si ricava $S=C/B/A=\#$ di set della cache= $128/16/4=2$, $XM=X/B$, $XS=XM\%S$, $XT=XM/S$:

```

=== T   X   XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF [SET]:TAG
=== R 124   7   3   1  12  0 [1]:3,0,0,0 [1]:0,0,0,0 [1]:3,-,-,-
=== W 169  10   5   0   9  0 [0]:3,0,0,0 [0]:0,0,0,0 [0]:5,-,-,-
=== R 167  10   5   0   7  1 [0]:3,0,0,0 [0]:0,0,0,0 [0]:5,-,-,-
=== W 245  15   7   1   5  0 [1]:2,3,0,0 [1]:0,0,0,0 [1]:3,7,-,-
=== R 183  11   5   1   7  0 [1]:1,2,3,0 [1]:0,0,0,0 [1]:3,7,5,-
=== W 119   7   3   1   7  1 [1]:3,1,2,0 [1]:1,0,0,0 [1]:3,7,5,-
=== R 235  14   7   0  11  0 [0]:2,3,0,0 [0]:0,0,0,0 [0]:5,7,-,-
=== W 163  10   5   0   3  1 [0]:3,2,0,0 [0]:1,0,0,0 [0]:5,7,-,-
=== R 288  18   9   0   0  0 [0]:2,1,3,0 [0]:1,0,0,0 [0]:5,7,9,-
=== W 309  19   9   1   5  0 [1]:2,0,1,3 [1]:1,0,0,0 [1]:3,7,5,9
=== R 310  19   9   1   6  1 [1]:2,0,1,3 [1]:1,0,0,0 [1]:3,7,5,9
=== W 308  19   9   1   4  1 [1]:2,0,1,3 [1]:1,0,0,1 [1]:3,7,5,9
=== R 213  13   6   1   5  0 [1]:1,3,0,2 [1]:1,0,0,1 [1]:3,6,5,9 (out: XM=15 XT=7 XS=1 )
=== W 196  12   6   0   4  0 [0]:1,0,2,3 [0]:1,0,0,0 [0]:5,7,9,6
=== R 377  23  11   1   9  0 [1]:0,2,3,1 [1]:1,0,0,1 [1]:3,6,11,9 (out: XM=11 XT=5 XS=1 )
=== W 166  10   5   0   6  1 [0]:3,0,1,2 [0]:1,0,0,0 [0]:5,7,9,6
=== R 362  22  11   0  10  0 [0]:2,3,0,1 [0]:1,0,0,0 [0]:5,11,9,6 (out: XM=14 XT=7 XS=0 )
=== W 233  14   7   0   9  0 [0]:1,2,3,0 [0]:1,0,0,0 [0]:5,11,7,6 (out: XM=18 XT=9 XS=0 )
=== R 163  10   5   0   3  1 [0]:3,1,2,0 [0]:1,0,0,0 [0]:5,11,7,6
=== W 169  10   5   0   9  1 [0]:3,1,2,0 [0]:1,0,0,0 [0]:5,11,7,6
    
```

P1 Nmiss=12 Nhit=8 Nref=20 mrate=0.6 AMAT=28

Situazione finale cache:

[0]:5,11,7,6
 [1]:3,6,11,9

Lista blocchi uscenti:

(out: XM=15 XT=7 XS=1)
 (out: XM=11 XT=5 XS=1)
 (out: XM=14 XT=7 XS=0)
 (out: XM=18 XT=9 XS=0)

ESERCIZIO 3)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
add \$1, \$2, \$3	F	D	X	M	W																				
lw \$4,0(\$1)		F	---	---	D	X	M	W																	
add \$6, \$1, \$4					F	---	---	D	X	M	W														
lw \$5,0(\$1)							F	D	X	M	W														
bne \$6, \$5, L1								F	---	---	D	X	M	W											
nop											F	D	X	M	W										
lw \$4,0(\$1)												F	D	X	M	W									
add \$6, \$1, \$4												F	---	---	D	X	M	W							
lw \$5,0(\$1)															F	D	X	M	W						
bne \$6, \$5, L1																F	---	---	D	X	M	W			
nop																	F	D	X	M	W				

Senza Forwarding

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21			
add \$1, \$2, \$3	F	D	X	M	W																			
lw \$4,0(\$1)		F	D	X	M	W																		
add \$6, \$1, \$4			F	---	D	X	M	W																
lw \$5,0(\$1)					F	D	X	M	W															
bne \$6, \$5, L1						F	---	---	D	X	M	W												
nop									F	D	X	M	W											
lw \$4,0(\$1)										F	D	X	M	W										
add \$6, \$1, \$4										F	---	D	X	M										
lw \$5,0(\$1)												F	D	X	M	W								
bne \$6, \$5, L1													F	---	---	D	X	M	W					
nop																F	D	X	M	W				

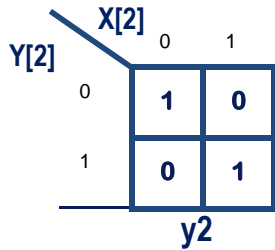
Forwarding

Senza forwarding: 25 cicli. Con forwarding 21 cicli. Nota: con forwarding, alla fine del ciclo 8 (e stessa cosa alla fine del ciclo 16) non e' possibile inoltrare il dato letto (stadio M di lw) in quanto, avendo 1-delay slot, la bne successiva consuma \$5 nello stadio D.

ESERCIZIO 4)

x	y	X	Y	q
+1	+1	001	001	1
+1	-1	001	111	0
-1	-1	111	111	3
-1	+1	111	001	2
+0	+0	000	000	1
+0	+1	000	001	1
+0	-1	000	111	0
+1	+0	001	000	1
-1	+0	111	000	2

X[2],Y[2] -> x2,y2
 00 -> 01
 01 -> 00
 10 -> 10
 11 -> 11

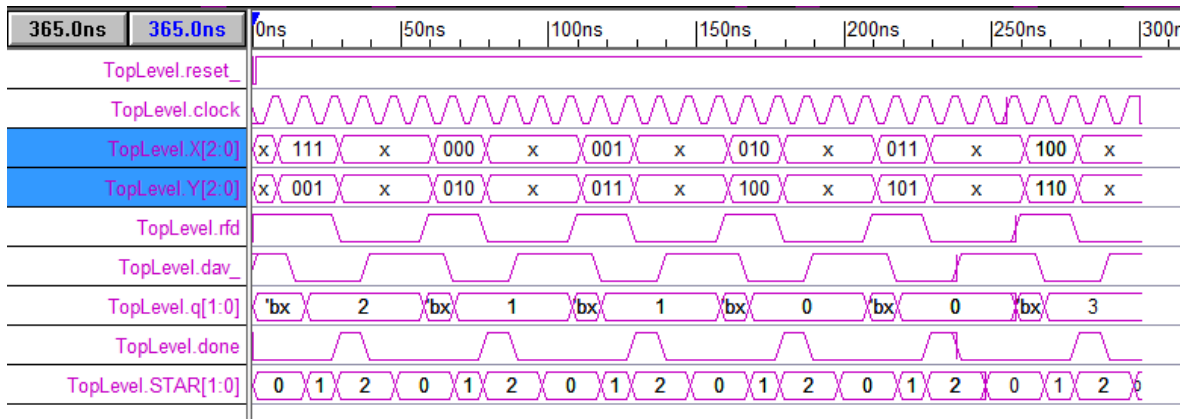


$$x2=X[2]$$

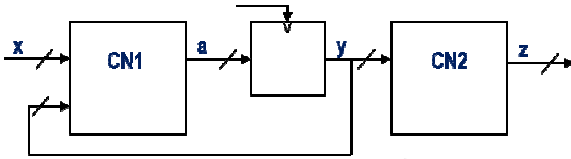
$$y2=X[2]/Y[2]+X[2]Y[2]$$

```

module XXX(rfd, dav_,X,Y, q,done, clock,reset_);
input clock, reset_;
output [1:0] q;
output rfd,done;
input dav_;
input [2:0] X,Y; wire y2_,x2_,p1,p2,y2;
reg [1:0] Q; assign q=Q;
reg RFD,DONE; assign rfd=RFD, done=DONE;
reg [1:0] STAR; parameter S0=0, S1=1, S2=2;
not n1(y2_,Y[2]);
not n2(x2_,X[2]);
and a1(p1,x2_,y2_);
and a2(p2,X[2],Y[2]);
or o1(y2,p1,p2);
always @(reset_==0) begin DONE<=0; RFD<=1; STAR<=S0; end
always @(posedge clock) if (reset_==1) #3
    casex (STAR)
        S0: begin RFD<=1; Q<={X[2],y2}; STAR<=(dav_==1)?S0:S1; end
        S1: begin RFD<=0; DONE<=1; STAR<=S2; end
        S2: begin DONE<=0; STAR<=(dav_==0)?S2:S0; end
    endcase
endmodule
    
```

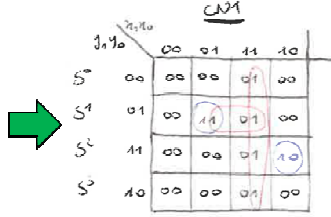


ESERCIZIO 5)



- 4 stati → STAR ha 2 bit
- → a ha 2 bit, y ha 2 bit
z ha 1 bit, (x ha 2 bit)

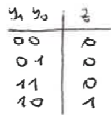
Stato	y1 y0
S0	00
S1	01
S2	11
S3	10



$$a_1 = \bar{x}_1 \bar{x}_0 \bar{y}_1 y_0 + x_1 \bar{x}_0 y_1 y_0$$

$$a_0 = x_1 x_0 + x_0 \bar{y}_1 y_0$$

a1, a0
CN2



$$z = y_1 \bar{y}_0$$

